SZÉCHENYI ISTVÁN EGYETEM
AUDI HUNGÁRI JÁRMŰMÉRNÖKI
KAR

BUILDING AN ELECTRIC DRIVE SYSTEM USING A BLDC MOTOR

FOR VEHICLE ENGINEER STUDENTS

Version: 1.1

Széchényi István University
Department of Power Electronics and Drives

1. Introduction

The project work is about building an electrical drive system using a BLDC motor, a motor driver and an Arduino Uno board. The motor driver takes care of supplying the motor with power and its control. The Arduino is used for higher level control of the drive according to the application that the drive system is used in. For example, in case of a drone application the Arduino would take care about flight control and generate the required rotational speed reference for the drive. It would feed this reference to the motor driver which would take care about driving the BLDC motor with the speed reference. In case of your project work you need to set the speed reference with a potentiometer.

In this document you will find some useful information and examples to help you with the compulsory project work. Based on the examples you should be able to do the project work. If this is not enough, feel free to contact us in email or at the laboratory classes.

The examples are implemented in "Tinkercad" environment. Tinkercad is a free and easy to use online application for 3D modelling, (basic) circuit simulation, coding and microcontroller-based application simulations. Before building the actual system in the laboratory you will need to implement it in a Tinkercad simulation.

To write and upload code on an Arduino board, you will need to use the Arduino IDE which you can download from the official website. The Arduino IDE (Integrated Development Environment) is a software application that serves as the primary platform for programming and developing projects on the Arduino platform. Using this link you can find some basic information and a starter guide for the IDE.

1. Electric drive systems

The block diagram of a general electric drive system can be seen on Fig 1.

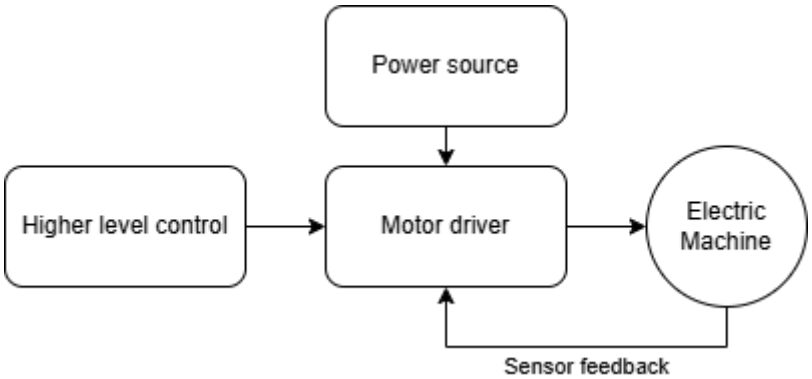

Fig.1. The block diagram of a general electric drive system.

A general electric drive system consists of a higher level controller which implements the control of the high level application of the drive, it generates the reference speed or torque that needs to be achieved by the electric machine in order to maintain the higher level application.

It is in most cases realized by a microcontroller. The complexity of this part is highly dependent on the application. In case of a very simple applications this can be omitted and replaced with some simple reference logic (pump, fan, conveyor belt).

The motor driver implements some kind of control strategy for the electric machine to achieve the reference value provided by the higher-level controller, and it converts the electrical energy supplied by the power source for the machine. A block diagram of a general 3 phase motor driver can be seen on Fig. 2.

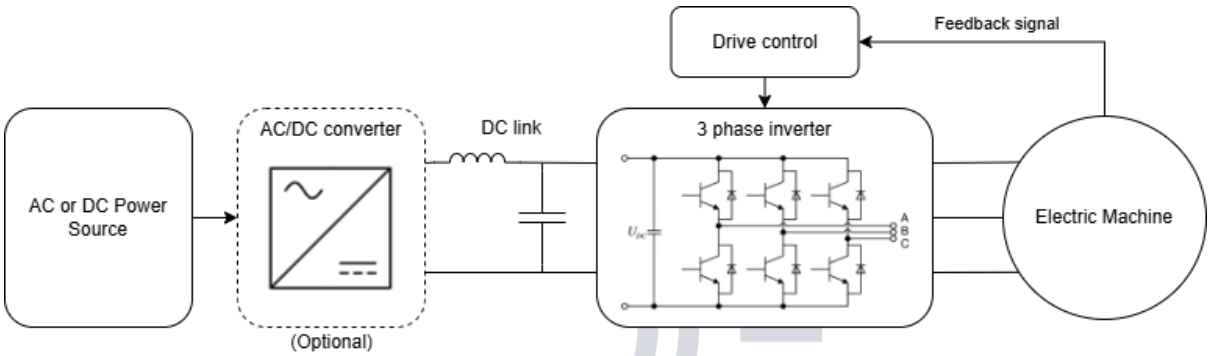

Fig.2. The block diagram of a general 3 phase motor driver.

2. The drive system you need to build using a BLDC motor

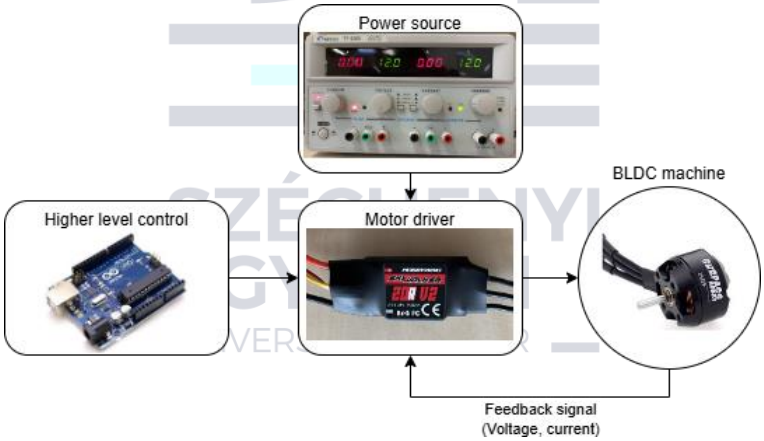

Fig.3. The block diagram of the BLDC drive system.

You can see the block diagram of the BLDC drive system with the actual components you need to use in Fig. 3.

You can find information about the used motor driver on the website of the manufacturer (There is a user manual under the support menu). The nominal continuous current of the controller is 20 A. The thicker red and black wires on the left side of the driver are for DC power input, you need to connect the DC power source here. It is designed to 3S-6S Li-po batteries, so for the project we will assume that we use a 3S battery, which means we apply 12V on the power supply (You can find more information about Li-po battery cells on this website). The thin red-black-white wires connected to a connector are for the control of the driver. Here the black goes to the GND pin of the Arduino, the red goes to the 5V of the Arduino and the white is for the PWM signal, controlling the speed reference. You will need to use a

potentiometer to set the speed reference for the drive (you will find more details on how to use a potentiometer with an Arduino in section 5). The thin yellow wire is for configuration of the drive, we don't need it for the project. On the right side of the driver there are three black wires, that are the output of the inverter of the driver, they go to the three phase of the motor.

The used BLDC motor is a Surpass hobby c2822 outrunner motor. It has a maximum current of 20A, which means the chosen motor driver is able to drive this motor. It is 1400KV which means it spins at 1400 RPM per volts on the stator. If you want to know more about BLDC motors and how to drive them watch this YouTube video.

3. Using Tinkercad

In order to use Tinkercad you need to create a free account, then after logging in you can create a new circuit design as you can see it in figure 4.

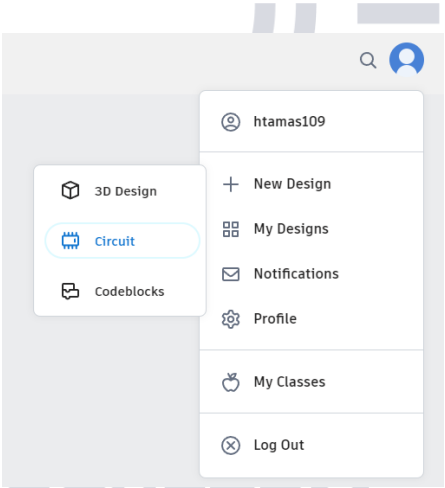

Fig.4. Creating new design

In a "Circuit design" you can place various electric components, sensors, actuators, microcontrollers, breadboards etc. from the "components" menu (at the right). You can connect the components using wires by clicking on the pins of the components. The editor displays realistic images of each component, which makes it easy to use as you can see it in figure 5.

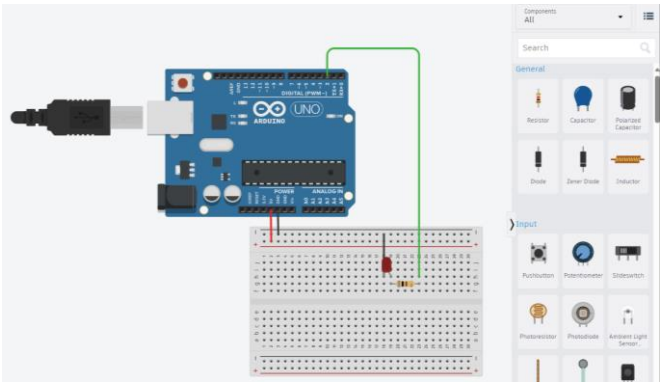

Fig.5. The user interface of Tinkercad

If your circuit contains a programmable device e.g., a microcontroller, you can write a code (by clicking on the “code” button on the top right) and upload it on the device. You can write your code using blocks or text, but you must use the text format as you will need to implement your written code on a real Arduino board using the Arduino IDE. By clicking the “Start Simulation” button the code gets compiled and uploaded to the selected microcontroller automatically and the simulation will start.

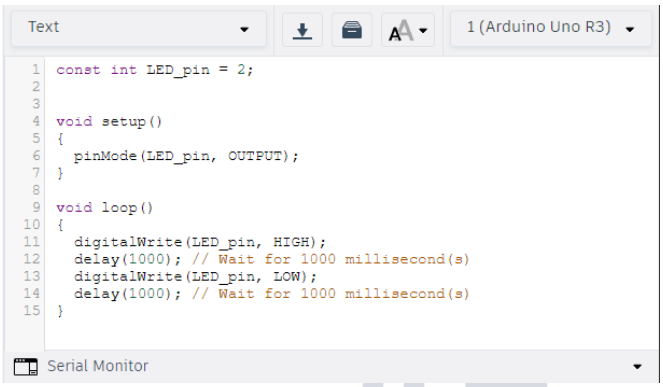

Fig.6. The code editor window

4. Modelling a BLDC drive in Tinkercad

In Tinkercad unfortunately there is no built in BLDC motor model and I cannot add any custom models, so I created an approximate model using built in components to emulate the BLDC motor with the motor driver. For creating the project in Tinkercad I will provide this part for you to use and you only need to create the reference signal to it using an Arduino. In the following section I will describe this drive model.

For the BLDC motor I only model its three phase stator winding using series RL circuits. Each phase of the winding can be represented by its DC resistance which gives the R part of the model and by its inductance which is the L part. The stator winding is connected in star (or Y). You can see the simple motor model on Fig. 7. The light bulbs in the model are to indicate the energized state of each phase of the model. Note that this is a very much simplified motor model. It does not consider any mechanical movements of the machine, it only models the stator winding without the rotor. Its purpose is only to visualize the rotating magnetic field generated by the motor drive.

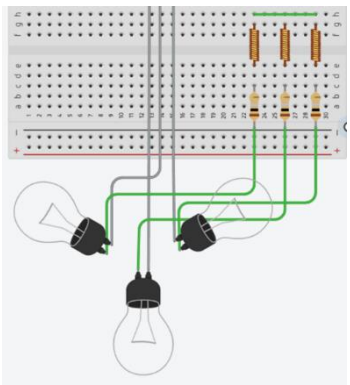

Fig.7. The BLDC motor model.

The motor driver is modelled using two main parts. The first main part is the three phase, two level, voltage source inverter, which can be seen in Fig. 8. An inverter is basically composed using some kind of semiconductor switches like MOSFETs or IGBTs. In case of a two level inverter there is two switches for every phase of the inverter. The high side inverter connects the positive DC voltage of the inverter to the output of the phase. The low side switch connects zero to the output.

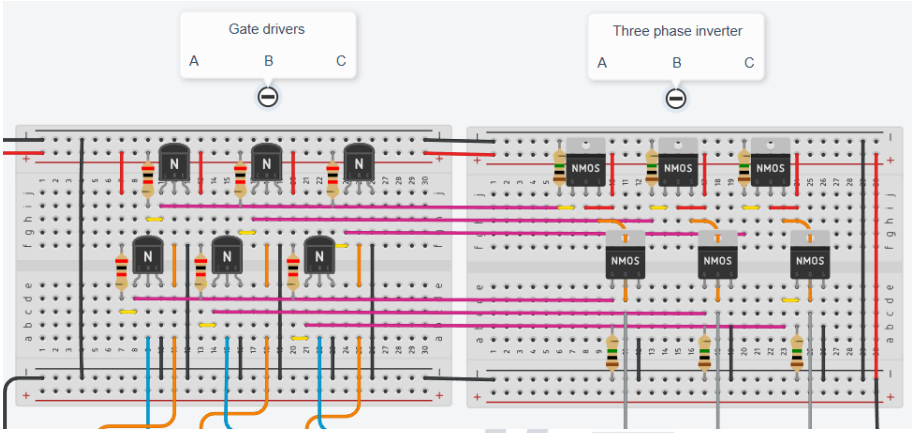

Fig.8. The three phase two level voltage source inverter in Tinkercad.

The BJTs on the left bread board are used as gate drivers for the MOSFETs in the inverter created on the right bread board. Gate drivers are used to turn on and off power MOSFETs with high frequency, because in high frequency switching operation there is significant current on the gate of this devices, which can not be provided directly by the digital pins on a microcontroller that controls the inverter. The schematics for an inverter like that is shown in Fig.9.

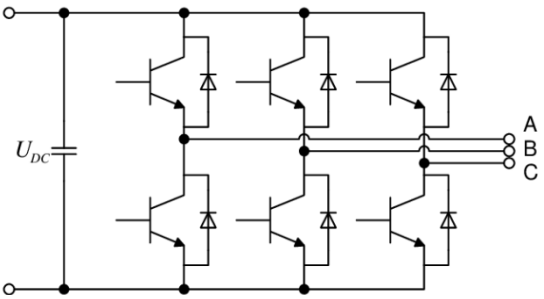

Fig.9. The schematics of a three phase two level voltage source inverter.

It is important to note that the two switches in a phase cannot be turned on at the same time. This means we can assign a binary value to every phase in the inverter. If this binary value is 1 the high side switch is on and the low side is off, and for 0 it's the other way around. In the case of a three phase inverter, we have 3 binary numbers which means we can have 8 different combinations, which gives 8 possible voltage vectors on the output of the inverter. The voltage vectors can be observed on Fig. 10. Where $v_1 v_2 v_4$are in line with phase a, b, c respectively, v_6 is given by turning on v_1 and v_2 at the same time, v_3 is given by v_2 and v_4, and v_5 is given by v_1 and v_4. If we turn on or off all three phases, we get v_0 and v_7 which are zero vectors.

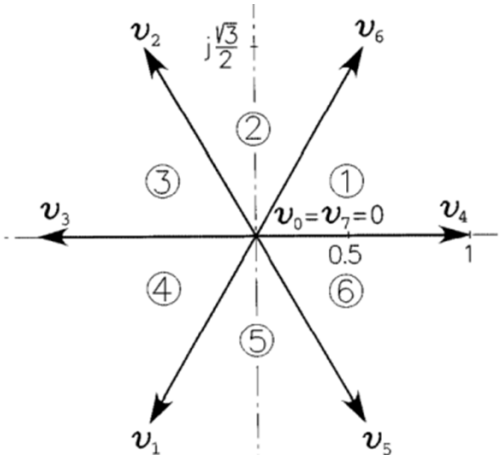

Fig.10. The 6 active and 2 zero voltage vector of the three phase, two level voltage source inverter.

The DC voltage source of the inverter is provided by a DC power supply, that can be seen in Fig. 11.

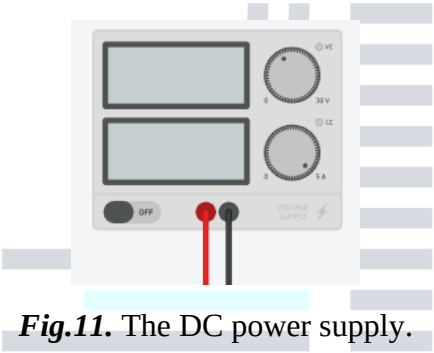

Fig.11. The DC power supply.

The second main part of the drive model is an Arduino that creates the control of the MOSFETs in the inverter (Fig. 12.).

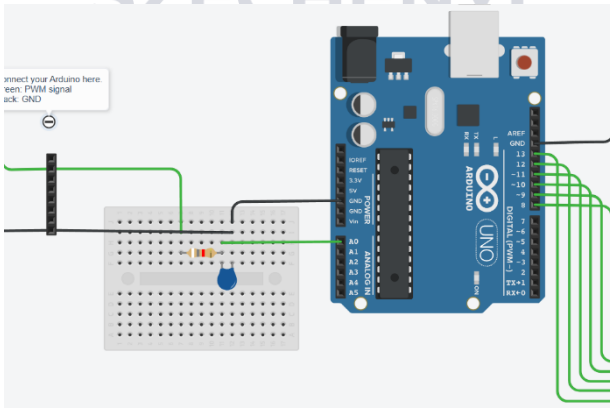

Fig.12. The Arduino controlling the inverter.

In a real drive system, the microcontroller would provide some control algorithm for the BLDC motor, for example V/f, FOC or DTC control. To implement any of these control algorithms we would need a more detailed motor model in the simulation, which we cannot acquire using the built-in components in Tinkercad. So, I only created the six-step control of the inverter generating the rotating magnetic field for the machine. This means iterating between the six active voltage vectors in order. This in itself would not be enough to drive a BLDC machine,

but it is enough to mimic the operation of the real drive system. The rotational speed of the generated magnetic field is proportional to the analog input the Arduino gets on A0, which you need to provide. The series RC circuit on A0 works as a low pass filter to create some stable voltage level form the PWM signal provided to the systems input by you. This voltage level is proportional to the duty cycle of the PWM signal so I can read it with A0 and have a value for the positive pulse width you provide.
In case of a real BLDC drive after power on you need to provide a full throttle (2 ms pulse width) wait a little (for example delay(3000)) and a zero throttle (1 ms pulse width) for the drive. This is to calibrate the input levels and also serves as a safety function so you can not power on the drive system and have the motor spin immediately. The drive gives you a sound feedback of this procedure using the motor. I also coded this functionality in the modelled drive system. The Arduino writes some messages about this on its serial monitor to provide information about its state. So its important to implement this sequence in the setup function of your code!

You can access the BLDC motor driver model using this link. Your task is to connect a new Arduino to the black 8 pin connector and write the code that controls the ESC modelled by the other one.

If you look into the code I created on the Arduino that controls the drive, you see that I control the pins of port B using registers. For this you can find an example at this link.

5. Using the analog pins on the Arduino

To measure analog signals microcontrollers, use analog to digital converters (ADC). The Arduino Uno has a 10-bit ADC, that can measure voltages between 0 and 5 V. This means it will give a digital value in the range of 0 – 1023 (2^10). This is called as a resolution which indicates the number of discrete values it can produce over the range of analog values. The Arduino Uno's ADC has 6 independent channel which means it can handle 6 analog input pins. If you want to learn more about ADC's click on the link. It is important to mention that in most cases microcontrollers can only measure voltage on their analog pins.
In order to access the converted digital value of an analog pin you can use the "analogRead()" function which returns the digital value of a specified analog input. You can see how to use it in the example project provided below.
To demonstrate how to use the analog inputs of an Arduino Uno I created a Tinkercad project, which measures the voltage of a potentiometer and sends the read value to the serial monitor.

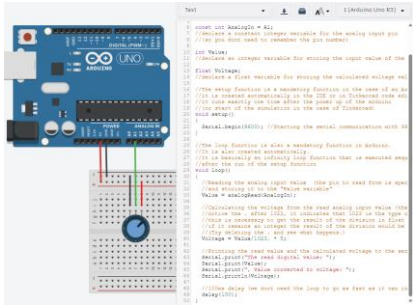

Fig.13. Measuring voltage on a potentiometer

Serial communication is a standardized communication protocol that is used in microcontrollers to communicate with the PC or with other devices. If you want to know more about serial communication click on the link. You can use the Arduino environment's built-in serial monitor (in Tinkercad too) to communicate with an Arduino board. Click the serial monitor button in the toolbar and select the same baud rate used in the call to begin(). If you want to learn more about Arduino's serial communication click the link.
The most important functions to use the serial monitor for the project work is "Serial.begin()" and "Serial.print()", "Serial.println()". The "Serial.begin()" function starts the serial communication between the board and the PC with the baud rate specified as the input of the function (for this project 9600 is fine). The "Serial.print()" and "Serial.println()" functions are used to print information to the serial monitor, they work the same way, except the second one prints a new line character to the end. You can see how to use them in the example. You can also send data from your PC to the Arduino using the serial monitor, but that is not needed for the project work.
(In Tinkercad you can open the serial monitor by clicking the "Serial Monitor" button at the bottom of the code window.)

You can access the Tinkercad design using this URL: Using Arduino's analogRead example. To open the design, you should click on the "tinker this" button. I provided comments to each line of the code as an explanation. It is strongly encouraged to play around in all the provided projects, try to understand the code, and write it on your own, also try other components etc.

6. Using the PWM pins on the Arduino

To control the rotational speed of the BLDC motor the motor driver needs to get a speed reference signal. This reference signal is a standardized pulse width modulated (PWM) signal. A PWM signal is a digital signal, which means that its value pulsates between a low (in most cases 0V) and a high (5V for an Arduino) value. As its name suggests, the width of the high pulses is modulated according to the information we want to represent. By changing the width of the high pulses, the duty cycle of the signal is changing, which is the ratio of the high and low pulses. For the motor driver we need to produce a PWM signal with a positive pulse width between 1 ms and 2 ms. A 1 ms positive pulse means 0 RPM for the motor and 2 ms means maximum speed.

The digital pins of the Arduino denoted with a tilde (~) sign are able to produce PWM signals. Microcontrollers use their built-in timer hardware to create PWM signals. If you want to know more about how it happens you can find more information using this link. In case of the Arduino IDE you can create PWM signals using the analogWrite() function. This function has two inputs, the first is the pin number on which you want to generate the PWM signal (you can use any one of the PWM compatible digital pins with a "~" sign next to them), and the second is an integer number between 0 and 255 to set the duty cycle of the signal. The Arduino board uses its timer peripherals to generate the PWM signal. The Uno has 3 timers called timer0, timer1 and timer2. When you use the analogWrite() function by default the output pins connected to timer0 and timer1 generate an approximately 490 Hz PWM signal and the ones connected to timer2 generate about 1 kHz PWM. For the project task you need an approximately 490 kHz signal so use pin 3,9,10 or 11. Also by default all of the timers are set to 8 bit, that's why you need to provide an integer value between 0 and 255 for the analogWrite() function, where 0 means a 0 ms positive pulse width and 255 means a positive signal for the full period time of the signal. A period time of a signal is the reciprocal of its frequency (in case of a 490 Hz signal the period time is 1/490 s). You can read more about Arduino Uno's timers on this site and see the difference between the pins connected to different timers on this Tinkercad example.
To adjust the duty cycle, use a potentiometer and the analogRead() function. Be careful that you need to provide a positive pulse width between 1 ms and 2 ms for the motor driver, which means you need to map the 0-1023 input value from the analogWrite() accordingly. You can do this with the map() function.